6800/ICCD

JOURNAL

INTERNATIONAL COMPUTER CENTER DIRECTOR

ICCD JOURNAL

The ICCD Journal is published four times yearly by the International Computer Center Director, P. O. Drawer 2790, Norman, Oklahoma 73070. The ICCD is a wholly owned division of Academic World Inc. Subscription rates are $18/year u.s. and Canada for membership, $21/year for countries outside the U.S. and Canada.

Opinions expressed by the authors are not necessarily those of the ICCD. Address all subscription correspondence to:

ICCD
P.O. Drawer 2790
Norman, Oklahoma 73070
405-364-1119

Articles are welcomed by readers of ICCD and high quality manuscripts. ICCD provides a free one-year membership to successful submitters.

PUBLISHER-EDITOR
Harold Zallen
ASSOCIATE EDITOR-GRAPHIC ARTS
Robert Checorski
COMPUTER TYPOGRAPHY
Fred Weddle

AN ACADEMIC WORLD INCORPORATED ENTERPRISE

Volume 1, No. 1 November, 1977

TABLE OF CONTENTS

page

SWTPC Graphics .... R. L. Smith 2

Biosin .... R. L. Hilbun 4

Personality: Evolution of a Complex Machine .... R. L. Hilbun 6

One Man's Opinion of Basic Timing Comparisons .... Mickey A. Ferguson WA4KDC 7

Megabaud 1.2 Maybe? .... R. L. Hilbun 9

Memtst—A Memory Program .... Emerson Brooks 10

The WA4KDC OS-II .... Mickey A. Ferguson 11

COMMENTS FROM THE PUBLISHER

This is the first of what we hope will be a long and healthy life for the ICCD. We believe that the ICCD Journal has a need which does not compete with existing journals, pamphlets, newsletters and magazines. As the issues become more sophisticated we feel the acceptance will grow. The 6800 and the successors to the 6800 are growing daily in popularity. The Newsletter name which was to be FIFO has been changed to the 6800 User's CHIP. It is here we will tell the happenings in the world of the 6800.

A special note of thanks to some of our well wishers—John Craig, Editor of Kilobaud and Mr. Dan Meyer, President of the Southwest Technical Products Corporation. Dan deserves a great deal of thanks for his support of the ICCD through our acquisition of test hardware. These tests and evaluations will be published from time to time.
We ask for advance notices to further support the needs of the SWTPC 6800 User who is not fortunate to have a computer store or a university in his immediate location. A note of what the future is to bring. More graphics, programs on billing-invoices and short programs from port to print. Evaluation of the Microware RT 68 MX is also underway. More on Discs. We are currently testing 2 sets of dual floppies and have 5 other dual floppy disc systems being used or at ICCD disposal. Perhaps an Operating System in DOS?

Now, what about you the reader-subscriber? What article do you have? What do you wish to share? Where are the General Ledgers with sub ledgers like the 8080 (Altair-MITS) Floppy Rom appearing in Interface Age for the 6800 Users? What about an interface for the 6502 PET with the 6800? We have the basis of fast turn-around time and will exercise this in future issues. The birth was quite painful and slow. Help us out with articles—full listings can be accomplished and we intend to document all programs possible. We ask you to tell us what you want, expect and need. Possible columns, if the need is expressed, there will be literature summaries, glitches, graphics, business systems, Letters to the Editor, etc. PLEASE TELL US. ICCD will act as your spokesperson in the 6800 world.

Harold Zallen, Ph.D.
Editor-Publisher

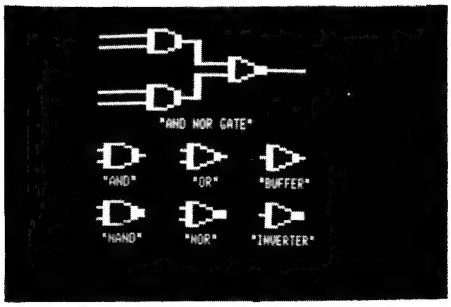
SWTPC GRAPHICS

In truth, it was the sight of the Enterprise stationed majestically above the Earth in a holding orbit that sold me on a SWTPC GT-6144 Graphic Terminal. It really does produce a very impressive Star Trek scene with the software provided by Southwest. But from that point on we are on our own. Assembly language programming on a extended basis gets old in a hurry besides leading to feelings of great personal inadequacy. My solution is this short program (written with liberal use of subroutines from the software supplied by Southwest) which allows the GT-6144 to be programmed with Technical Systems Consultants', MICRO BASIC Plus.©

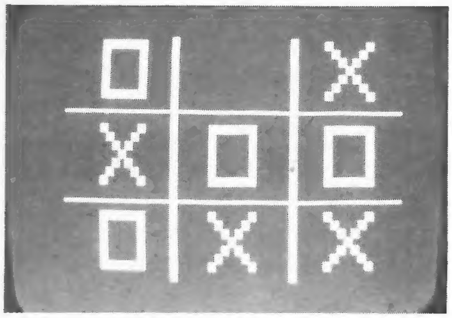
Access to the Graphic routine is made through the EXTERNAL statement. This causes a "branch to subroutine" which appears to the BASIC exactly like a GOSUB statement. Graphic in turn looks at the values stored in the "X", "Y" and "Z" variables for the GT-6144's control functions and dot position data. All functions of the Graphic Terminal are under software control and available to BASIC, including a full screen erase. The terminal's display is formed from a 64 by 96 dot matrix. Positioning is determined by specifying the number of dot positions to the right and up from the lower left hand corner of the screen. Horizontal placement is the value stored in the "X" variable and has a valid range of from 0 to +63. Vertical position comes from the "Y" variable and its range is from 0 to +95.

"Z" is used for the control functions and are given in the program listing. Before entering Graphic through EXT the proper values must be present in these variables. One disadvantage to using BASIC with the Graphic Terminal is a loss in speed. A fast moving "Pong" type game might be difficult to make using BASIC.

```
PAGE 001   GRAPHIC
                         NAM    GRAPHIC

                   * * * * * * * * *
                   *GRAPHICS PROGRAM
                   *     FOR
                   *TSC MICROBASIC +
                   *     AND
                   *SWTPC   GT-6144
                   *
                   * 4/77  R L SMITH
                   *
                   *PROGRAM ENTRY THRU
                   *'EXT' COMMAND
                   *BASIC VARIABLES
                   *'X','Y' AND 'Z'
                   *ARE USED.
                   *'DOT' POSITION
                   *DETERMINED FROM
                   *VALUES STORED IN
                   *'X'(HORIZ) 0-63
                   *-LEFT TO RIGHT
                   *    AND
                   *'Y'(VERT) 0-95
                   *-BOTTOM TO TOP
                   *
                   *'Z' IS USED FOR
                   *CONTROL FUNCTIONS
                   *Z=0  WRITE
                   *Z=1  REVERSE SCREEN
                   *Z=2  NORMAL SCREEN
                   *Z=3  TVT OFF
                   *Z=4  TVT ON
                   *Z=5  GRAPHICS ON
                   *Z=6  GRAPHICS OFF
                   *Z=7  WRITE
                   *Z=8  DOT DELETE
                   *Z=9  FULL SCREEN
                   *     ERASE
                   *
                   * * * * * * * * *
                   *
                   *
                   *
                         OPT    O,S,NOP

      00F7         INX   EQU    $00F7
      00FA         INY   EQU    $00FA
      00FD         INZ   EQU    $00FD
      E1D1         OUTEEE EQU   $E1D1
      8014         PIAS  EQU    $8014
1F00                     ORG    $1F00
1F00 36            START PSH A

1F01 37                  PSH B

1F02 FF 1FAC             STX    INDX
1F05 D6 FB               LDA B  INZ-2
1F07 DB FC               ADD B  INZ-1
1F09 DB F5               ADD B  INX-2
1F0B DB F6               ADD B  INX-1
1F0D DB F8               ADD B  INY-2
1F0F DB F9               ADD B  INY-1
1F11 5D                  TST B

1F12 26 34               BNE    EXIT
1F14 8D 38               BSR    PIASET

1F16 D6 FD               LDA B  INZ
1F18 C1 09               CMP B  #9
1F1A 27 4B               BEQ    ERASE
1F1C 5D                  TST B

1F1D 27 0B               BEQ    WRITE
1F1F C1 07               CMP B  #7
1F21 22 07               BHI    WRITE
1F23 86 DF               LDA A  #$DF
```




```
1F25 1B                  ABA

1F26 8D 32               BSR    OUTCH
1F28 20 1E               BRA    EXIT
1F2A 96 F7         WRITE LDA A  INX
1F2C 81 63               CMP A  #$63
1F2E 22 18               BHI    EXIT
1F30 8D 64               BSR    BCDBIN

1F32 D6 FD               LDA B  INZ
1F34 C1 08               CMP B  #8
1F36 27 02               BEQ    DELETE

1F38 8B 40               ADD A  #$40
1F3A 8D 1E        DELETE BSR    OUTCH
1F3C 96 FA               LDA A  INY
1F3E 8D 56               BSR    BCDBIN

1F40 16                  TAB

1F41 86 5F               LDA A  #$5F
1F43 10                  SBA

1F44 8B 80               ADD A  #$80
1F46 8D 12               BSR    OUTCH
1F48 33           EXIT   PUL B

1F49 32                  PUL A

1F4A FE 1FAC             LDX    INDX
1F4D 39                  RTS

                  *PIA SETUP
1F4E CE 8014      PIASET LDX    #PIAS
1F51 C6 FF               LDA B  #$FF
1F53 E7 00               STA B  0,X
1F55 C6 3F               LDA B  #$3F
1F57 E7 01               STA B  1,X
1F59 39                  RTS

                  * OUTPUT ROUTINE
1F5A A7 00        OUTCH  STA A  0,X
1F5C C6 37               LDA B  #$37
1F5E E7 01               STA B  1,X
1F60 E6 00               LDA B  0,X
1F62 C6 3F               LDA B  #$3F
1F64 E7 01               STA B  1,X
1F66 39                  RTS

                  *FULL ERASE
1F67 B6 1FAA      ERASE  LDA A  HORZ
1F6A 81 40               CMP A  #$40
1F6C 27 1E               BEQ    OUT
1F6E BD 1F5A             JSR    OUTCH
1F71 B6 1FAB      INCREM LDA A  VERT
1F74 81 60               CMP A  #$60
1F76 27 0A               BEQ    SPEC
1F78 8B 80               ADD A  #$80
1F7A BD 1F5A             JSR    OUTCH
1F7D 7C 1FAB             INC    VERT
1F80 20 EF               BRA    INCREM

1F82 86 00        SPEC   LDA A  #0
1F84 B7 1FAB             STA A  VERT
1F87 7C 1FAA             INC    HORZ
1F8A 20 DB               BRA    ERASE
1F8C 86 00        OUT    LDA A  #0
1F8E B7 1FAA             STA A  HORZ
1F91 B7 1FAB             STA A  VERT
1F94 20 B2               BRA    EXIT
                  *PACKED BCD TO
                  *BINARY CONV.
1F96 B7 1FA9      BCDBIN STA A  TEMP
1F99 84 F0               AND A  #$F0
1F9B 44                  LSR A

1F9C 16                  TAB

1F9D 54                  LSR B

1F9E 54                  LSR B

1F9F 54                  LSR B

1FA0 1B                  ABA

1FA1 1B                  ABA

1FA2 F6 1FA9             LDA B  TEMP
1FA5 C4 0F               AND B  #$0F
1FA7 1B                  ABA

1FA8 39                  RTS

1FA9 00           TEMP   FCB    0
1FAA 00           HORZ   FCB    0
1FAB 00           VERT   FCB    0
1FAC 0000         INDX   FDB    0
                         END

INX    00F7
INY    00FA
INZ    00FD
OUTEEE E1D1
PIAS   8014
START  1F00
WRITE  1F2A
DELETE 1F3A
EXIT   1F48
PIASET 1F4E
OUTCH  1F5A
ERASE  1F67
INCREM 1F71
SPEC   1F82
OUT    1F8C
BCDBIN 1F96
TEMP   1FA9
HORZ   1FAA
VERT   1FAB
INDX   1FAC

TOTAL ERRORS 00000
```



```
5000 REM      TIC-TAC-TOE
5005 REM GRAPHIC DISPLAY
5010 REM SUBROUTINES WRITTEN IN
5020 REM TSC MICRO BASIC PLUS WITH
5030 REM GRAPHIC AND SWTPC CT-6144
5040 REM
5050 REM 5/77        R L SMITH
5100 REM GAME FIELD
5105 REM ********************
5110 Z=0
5120 Y=31:FOR X=5 TO 55:EXT:NEXT X
5130 Y=62:FOR X=5 TO 55:EXT:NEXT X
5140 X=20:FOR Y=5 TO 90:EXT:NEXT Y
5150 X=38:FOR Y=5 TO 90:EXT:NEXT Y
5160 RETURN
5170 REM ************************
5180 REM 'X' SUBROUTINE
5190 U=6+((U-1)*30):L=9+((L-1)*17)
5200 Y=U+20:FOR X=L TO L+6
5210 EXT:Y=Y-1:EXT:Y=Y-1:EXT:Y=Y-1
5220 NEXT X
5230 Y=U
5240 FOR X=L TO L+6:EXT:Y=Y+1
5250 EXT:Y=Y+1:EXT:Y=Y+1:NEXT X:RETURN
5260 REM '0' SUBROUTINE
5265 REM ********************
5270 U=6+((U-1)*30):L=9+((L-1)*17)
5280 FOR Y=U+20 TO U+18 STEP-1
5290 FOR X=L TO L+6:EXT
5300 NEXT X:NEXT Y
5310 X=L+6:FOR Y=U+20 TO U STEP-1
5320 EXT:NEXT Y
5330 FOR Y=U TO U+2
5340 FOR X=L+6 TO L STEP-1
5350 EXT:NEXT X:NEXT Y
5360 X=L:FOR Y=U TO U+20:EXT:NEXT Y
5370 RETURN
```

The program is located at 1F00 through 1FAD. Entry is at 1F00. With the exception of the temporary storage area all addressing is in the relative mode making relocation a simple matter. Of course, if the Graphic is relocated this change should also be reflected in the EXTERNAL jump of the TSC BASIC. Graphic assumes the GT-6144 PIA is in Port Number 5, 8014 (HEX), it may be changed by placing a different port address at 1F4F and 1F50.

R. Lynn Smith
PSC No. 1 Box 7327
APO San Francisco, California
96286

HUMPS AND SUMPS

(or How To Know When To Grit Your Teeth)

Everything has its ups and downs, your mother told you that! But it is more real than you may realize. Repetitive cycles occur in many systems; day and night, seasons, sunspots and whatever it is that makes girls different from boys. Ah, now that we're close to home, what are the cyclic rhythms that are a part of our physiological make-up? Other than the afore (un) mentioned, there is the circurian rhythm. This interpretation of your bodily functions has been well studied by scientists the world over. It basically describes your subtle physiological changes during each day. The circurian rhythm is demonstrated by such things as afternoon sleepiness and jet lag. The application of this rhythm is quite good for the relief of guilt over drinking alcohol during lunch or on the plane.

On to bigger cycles! The biorhythm is a physiological cycle, which covers a larger time interval. Due to this fact, scientific studies have been based on statistics and is therefore less definitive than circurian rhythms. But as the statistics pile up, it appears that this longer cycle describes emphatic changes in your daily life.

The biorhythm consists of three curves usually plotted as sine waves. The shortest is the physical curve which completes its cycle in 23 days. Next is the emotional curve which is 28 days long and finally the intellectual which appears to run 33 days. It is

```
    BIOSIN:BIRTH ICCD
   BIRTH DATE:SATURDAY,10 1 77
   START DATE:SATURDAY,10 15 77

       DOWN              CRITICAL
10 15         P              X         I
10 16     P             E    *     I
10 17  P           E         * I
10 18 P       E            I *
10 19 P    E          I      *
10 20   X         I          *
10 21 E     P I              *
10 22 E    I     P           *
10 23 E  I             P     *
10 24  IE                    X
10 25 I    E                 *     P
10 26 I       E              *          P
10 27 I            E         *               P
10 28   I               E    *                   P
10 29     I                  X                     P
10 30        I               *    E                P
10 31           I            *         E          P
11 1                I        *             E   P
11 2                     I   *             P  E
11 3                         X        P          E
11 4                         *  PI                 E
11 5                      P  *        I
11 6                P        *            I         E
11 7          P              *               I    E
11 8      P                  *                 EI
11 9   P                     *              E     I
11 10 P                      *         E            I
11 11 P                      *    E                 I
11 12   P                    X                      I
11 13       P           E    *                      I

    P-PHYSICAL
    E-EMOTIONAL
    I-INTELLECTUAL
```

curious to note that the emotional and physical curves were determined simultaneously and independently by two European physicians in the early 1900's, Dr. Willhelm Fleiss of Berlin and Dr. Hermann Swoboda of Vienna. Both gentlemen base their theories on observed physical and physiological changes. Dr. Fleiss believed the emotional curve occurred in women and the physical in men. When it was observed that both curves are represented in men and women, it lended itself to support certain theories of Dr. Fleiss' friend and patient, Sigmund Freud, the considered founder of modern day psychiatry.

The intellectual curve was observed later by Dr. Alfred Teltscher, an engineering professor in Innsbruck. Teltscher's work indicated fluctuations in test scores that could be related back to the students birthdate.

The basic theory states that the curves start at zero and are in phase at birth—(a good starting point). The progression from that point is regular and therefore easy to calculate.

With each of the three curves, there are three parts: UP, DOWN and CRITICAL. Consider these curves to repesent a stress or

energy cycle. During the up portion, you are dissipating energy and may be able to accomplish more or be in a better state of mind than in the down portion where you are in what may be called a "recharging state". The transition between these two states appears to be of most significance. At time zero, stress statistics state that we are more accident prone, therefore, it is termed the "critical" period.

The program is written to plot biorhythm curves for birthdates and starting dates within the twentieth century. The dates should be entered as 9,26,47 for September 26, 1947. The day of the week for the two entered dates will be printed. The three curves are represented by characters. Where two characters need to occupy the same space an "X" is printed. There is no significance to the presence of an"X". Critical days may or may not have an "X" in the center column. Visually they form a curve of like letters. When the curve crosses the critical line that day is to be considered critical.

The program will initially ask for the port number and column width. This will allow you to output the curve to any output device. If the column width is greater than 64 characters or so, you might want to add:

0X845 PRINT

This should make the curve more readable. Upon the completion of each curve, information is requested for the next curve. The program must be re-started to change the output port and column width.

```
0010 REM *BIOSIN BY R.L.HILBUN
0020 REM *WRITTEN FOR ICCD
0030 REM
0040 REM *ENTER BASE DATA
0050 DIM N(12),S(12),D$(7)
0060 DATA 31,28,31,30,31,30,31,31,30,31,30,31
0070 DATA 0,31,59,90,120,151,181,212,243,273,304,334
0080 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY
0090 FOR I=1 TO 12
0100 READ N(I)
0110 NEXT I
0120 FOR I=1 TO 12
0130 READ S(I)
0140 NEXT I
0150 FOR I=1 TO 7
0160 READ D$(I)
0170 NEXT I
0180 REM *DEFINE FUNCTIONS
0190  P1=2*3.14159
0200 DEF FNP(X)=INT(A*SIN(P1*X/23)+.5)
0210 DEF FNE(X)=INT(A*SIN(P1*X/28)+.5)
0220 DEF FNI(X)=INT(A*SIN(P1*X/33)+.5)
0230 REM *HOME UP & ERASE PAGE (CT-1024)
0240 PRINT CHR$(16);CHR$(22);
0250 REM *SET PRINTER PARAMETERS
0260 INPUT "OUTPUT PORT",P
0270 INPUT "COLUMN WIDTH",W
0280  A=(W-8)/2
0290  C=A+7
0300 PRINT CHR$(16);CHR$(22);
0310 REM *ENTER INDIVIDUAL DATA
0320 PRINT "ENTER DATE AS 4,1,77 FOR"
0330 PRINT " APRIL 1,1977"
0340 PRINT
0350 INPUT "NAME",N$
0360 INPUT "DATE OF BIRTH",M1,D1,Y1
0370 INPUT "STARTING DATE",M2,D2,Y2
0380 INPUT "NUMBER OF DAYS TO PLOT",L
0390  M=M1
0400  D=D1
0410  Y=Y1
0420 GOSUB 1220
0430  T1=T
0440  B$=D$
0450  M=M2
0460  D=D2
0470  Y=Y2
0480 GOSUB 1220
0490  T2=T
0500  S$=D$
0510 REM *FIND AGE IN DAYS
0520  T0=T2-T1
0530 REM *START COUNT
0540  L1=1
0550 PRINT CHR$(16);CHR$(22)
0560 PORT= P
0570 REM *PRINT HEADING
0580 PRINT TAB(4);"BIOSIN:";N$
0590 PRINT TAB(4);"BIRTH DATE:";B$;",";M1;D1;Y1
0600 PRINT TAB(4);"START DATE:";S$;",";M2;D2;Y2
0610 PRINT
0620 PRINT TAB(8);"DOWN";TAB(11+(W-22)/2);"CRITICAL";TAB(W-3);"UP"
0630 REM *PRINT DATE
0640 PRINT M2;D2;
0650 REM *CORRECT DATA FOR LEAP YEAR
0660 IF Y2/4=INT(Y2/4) THEN N(2)=29
0670 REM *INCREMENT DAYS & MONTHS
0680  D2=D2+1
0690 IF D2<=N(M2) THEN 740
```

After you run your chart, don't expect it to tell you your good and bad days, but rather observe your potentials in relation to the curves. Some adjustment may be required to obtain a chart so that your observations will coincide, don't hesitate to do this! Once a "corrected" birthday is fixed, the cyclic patterns will be as described in the program and then maximum benefit from the chart can be obtained. These corrections are usually less than four days.

```
0700  D2=1
0710  M2=M2+1
0720 IF M2>12 THEN M2=1
0730 REM *CALCULATE SYMBOL POSITION
0740  F(1)=C+FNP(TO)
0750  F$(1)="P"
0760  F(2)=C+FNE(TO)
0770  F$(2)="E"
0780  F(3)=C+FNI(TO)
0790  F$(3)="I"
0800  F(4)=C
0810  F$(4)="*"
0820 REM *PLACE SYMBOLS IN ORDER
0830 FOR I=1 TO 3
0840 FOR J=I+1 TO 4
0850 IF F(I)<F(J) THEN 970
0860 IF F(I)<>F(J) THEN 910
0870  F(I)=0
0880  F$(I)=""
0890  F$(J)="X"
0900 GOTO 970
0910  Q=F(I)
0920  Q$=F$(I)
0930  F(I)=F(J)
0940  F$(I)=F$(J)
0950  F(J)=Q
0960  F$(J)=Q$
0970 NEXT J
0980 NEXT I
0990 REM *PRINT SYMBOLS
1000 FOR I=1 TO 4
1010 PRINT TAB(F(I));F$(I);
1020 NEXT I
1030 PRINT
1040 REM *INCREMENT AGE
1050  TO=TO+1
1060 IF L=L1 THEN 1100
1070 REM *INCREMENT COUNT
1080  L1=L1+1
1090 GOTO 640
1100 PRINT
1110 PRINT TAB(5);"P-PHYSICAL"
1120 PRINT TAB(5);"E-EMOTIONAL"
1130 PRINT TAB(5);"I-INTELLECTUAL"
1140 IF P=1 GOTO 300
1150 REM *FORM FEED (PR-40)
1160 FOR I=1 TO 12
1170 PRINT
1180 NEXT I
1190 PORT= 1
1200 GOTO 300
1210 REM *CALCULATE NO. DAYS SINCE YEAR              1900
1220  T=365*Y+INT((Y-1)/4)+S(M)+D
1230 IF Y/4<>INT(Y/4) THEN 1260
1240 IF M>2 THEN T=T+1
1250 REM *CALCULATE DAY OF WEEK
1260  K=INT(((T/7)-INT(T/7))*7+.5)+1
1270  D$=D$(K)
1280 RETURN
```

R. L. Hilbun is a graduate in engineering from Mississippi State University and is presently District Production Engineer with Universal Resources Corporation, Oklahoma City, Oklahoma. His interest in Psychology and Physiology was developed while pursuing graduate studies in Psychology at McNeese State University, Lake Charles, Louisiana.

R.L. Hilbun
8109 Willow Creek Blvd.
Oklahoma City, Oklahoma 73132

PERSONALITY: EVOLUTION OF A COMPLEX MACHINE?

Behavior is the output of personality. To form theories of personality, many great men and women have closely observed the behavior of others and themselves. Although most theories have been advanced within the last one hundred years, the sum of the ideas presented gives a very complete picture of the character of the "Human Machine."

During the last few decades, science of electronics also has advanced. The paralleled development of these two sciences can now merge and in the author's opinion, our knowledge of electronics whch is very complete and exact, can be used to form a variety of models to help us understand the innate make up of our various personalities. Even without making electronic models to represent our human make up, much can be learned by observing the devices currently in use in the light of how they might be like our own function but in vastly simpler form.

There is little magic in electronics. The same computers in use today could be mechanical or fluidedic. It is easy to see that electrical devices offer much less design and construction problems and greater speed. Another form of construction which could be used is some combination of an electrochemical design. This is the general type of device that each of us have within our own body and more specifically our crainal cavity.

Most electronic devices are of the dedicated type; they perform one function with no method of self modification. These simple designs can be compared to plant life and the limited variation from the norm. The higher order of devices are not as dedicated. We commonly term these as computers. The devices have characteristics or if you wish, "personalities" and by applying some imagination can be crudely used to help us understand our own personalities.

Computer is a broad term, but let us say that any device with a large number of logic circuits and a memory is usable for study. Devices of this type, no matter what their particular use are good for subject models to form this author's theory of personality.

The birth of a person and the powering up of a computer are much the same. The basic idiosyncrasies, heredity and archetypes, are displayed. With a computer the display is instant; the "garbage" that is displayed is usually the same with each power-up.

Just as electricity follows the path of least resistance, so do our behaviors reflect the sum total of all our motivations. The young child and the unprogrammed machine merely try to reach a state of equilibrium. As the memory is filled both human and machine modify their behavior. In both cases the addition to the memory can 1) come from the environment; 2) be generated from within; or 3) a combination of the two. The latter case for the human could be subjective interpretation.

It appears that our memory may be very machine-like. One physiological experiment involved electrical excitation of various parts of the human brain. he subjects reported a very vivid recall of some past experience. The experience was in most cases an ordinary one, that is, not a traumatic incidence as would be expected. If in fact, the human brain stores most of the information that it receives, it would be safe to say that one human brain stores as much information as all the electronic devices now in existence!

Most present day computers have by nature total recall ability for information that has not been discarded and even though the data is manipulated, little if any, interpretation is involved. The human, in this respect, is vastly different. Firstly, verbatum recall is the exception rather than the rule. Most recalled information is recalled as an interpretation. This same recalled information is determining that person's behavior. It can be said then that the interpretation of partially recalled information plus the basic initial make-up of a person determines his personality.

This barely touches on the reasons for various types of personalities or the general make-up of a single personality, however, the correlation between our information processing machines in our heads and in our computer centers is, in my opinion, becoming increasingly important. Even though our knowledge of electronics is exact, the builders of some of the more complex computers have been unable to explain what appears to be subjective interpretation taking place in their machines.

Ron L. Hilbun
8109 Willow Creek Blvd.
Oklahoma City, Oklahoma 73132

ONE MAN'S OPINION OF BASIC TIMING COMPARISONS

Since the article on BASIC timing comparisons appeared in Kilobaud (1) many people have asked my thoughts on this subject. Most are rather surprised to find that I am not worried about 6800 BASIC being quite slow - in fact, I could care less. I have been asked how to get Altair 680 BASIC (without paying MITS an arm and a leg for it). I have been asked to rewrite OSI's 6502 Basic for the 6800. My answer to both of these questions was,"Why? It's not worth the effort.".

Because there is apparently considerable interest in the subject, I decided to look into the relative speeds of the various BASIC's. SWTPC's 4K BASIC "feels" faster than their 8K BASIC, so I ran the "benchmark" programs from the Kilobaud article using SWTPC 4K BASIC. To my surprise, there was virtually no difference between the times listed in the Kilobaud article for SWTPC 8K BASIC V1.0 and those I obtained with my system using 4K BASIC. Actually, I shouldn't have been surprised as both use Binary Coded Decimal (BCD) arithmetic and the same precision. Then I ran the benchmark programs using SWTPC 8K BASIC V2.0 (which is supposed to be faster than version 1.0). Again, I found no difference in speed!

Shortly thereafter I had the opportunity to ask Dr. Robert Suding of the Digital Group about

his feelings on the Kilobaud article. You may be surprised to know that he was not the least upset that the Digital Group's Maxi-BASIC could only manage to tie for seventh fastest when it was used in a Z-80 system running at full speed. His response was essentially the same as that of Robert Uiterwyk in the addendum to the Kilobaud article. The facts of BCD arithmetic and additional precision again appeared. This led me to run the benchmark programs using Robert Uiterwyk's Micro-BASIC, which uses double byte integer math. The results were as follows:

Benchmark Number	Time (in seconds)
1	1.4
2	8.25
3	15.25
4	15.9
5	21.6
6	30.5
7	46.3

If you compare this to the chart in the Kilobaud article, you will find this 6800 BASIC is somewhat slower than APPLE BASIC© (2) but faster than every other BASIC tested on a microcomputer! Before you run out and get a copy of Mr. Uiterwyk's Micro-Basic, think about what you can do with it. The answer is: **very little**. It has limited statements and commands, very limited math range, integer math only, etc. I seriously doubt whether you would be willing to trade 4K BASIC for it (much less 8K BASIC) just to get the additional speed.

What is the answer to speeding up BASIC? I considered interfacing a calculator chip to my 6800. Surely this would be the answer! Modify BASIC to have BASIC give the calculator chip the problem then pick up the answer from the calculator chip after it had computed the solution. But, how much time would be gained by using this method? To aid in determining the answer to this question, I wrote a simple program in BASIC to do factorials. I did this assuming I could compare the time required by BASIC vs. the time required by my scientific calculator and obtain an approximate answer. The factorial program is as follows:

```
 10 INPUT "TYPE A NUMBER
(69 or less)", N
 20 F=1
 30 N=ABS(N)
 40 IF N 69 THEN STOP
 50 IF N=0 THEN 100
 60 FOR X=N TO 1 STEP -1
 70 F1=F*X
 80 F=F1
 90 NEXT X
100 PRINT "THE FACTORIAL
OF ";N;" is ";F
110 END
```

The results of this comparison were so astounding that I re-ran the comparison many times. My scientific calculator required approximately twice the time required by BASIC to arrive at the answer! Now I know that interfacing a calculator chip to the 6800 won't speed things up but would actually slow BASIC down! I am now of the opinion that you would have to build a TTL math board which would run on its own clock at a much higher speed than the 6800 just to gain a few seconds. If you decide to go this route, I wish you well; because you're in for the hassle of your life! And don't be surprised if you only speed BASIC up very slightly.

Why would you only get a slight increase in speed by using a TTL math board running at, say, 30 MHz? You would get a terrific increase in the speed of addition, subtraction, multiplication, and division; but BASIC spends only a small portion of its time in these operations. You would not speed up things such as
FOR . . . . NEXT loops. Consider for a moment the simple loop:

```
10 FOR X=1 TO 100
10 NEXT X
```

```
10 FOR X=01 TO 03
20 NEXT X
```

Remember, when you are 'talking' to BASIC, you are using ASCII; which means that a 1 is a 31 and 100 is 31 30 30. In order for BASIC to use this information, it must convert it (each and every time it is encountered in your program) to packed BCD format. Sounds simple enough, right? No problem to convert 31 to 1. True enough, if BASIC could do it that way, which it can't because of its precision. With the simple loop shown above, you are effectively telling BASIC the following:

If you include a STEP factor, you slow the conversion and execution even more. You can see from the above example that BASIC must worry about the sign of the number, the number itself (to nine decimal places), the sign of the exponent, and the exponent itself; even when it is dealing with the simple number 1!

Altair Basic may be the answer for you if you are willing to settle for 50% less precision, unexplained intermittent loss of precision (3), etc. Personally, I would rather wait a little longer and **know** the answer to the problem given to BASIC is correct. Why am I unconcerned that 6800 BASIC is slow? All of the current crop of BASIC's for **all** of the microprocessors are slow! This is because they are interpreters and interpreters are slow. Your BASIC is a machine language program that thinks it is a computer. So, your 6800 is running a machine language program (BASIC) which thinks it is a computer running your BASIC program. The result is slow, very slow. All interpreters suffer this lack of speed and all are slow.

How would you like your BASIC programs to run hundreds or even thousands of times faster than

they do now? Easy, write them in machine language. Too much trouble? Well, why not let your 6800 write them in machine language for you? This is what a compiler does. It converts your programs from BASIC (or other high level language) to machine language. By the time you read this, you should be able to spend about $50 and get a compiler for your 6800 that will do everything that 8K BASIC will do (and more). And think about it, your programs will run thousands of times faster! An additional benefit of compiling your programs is that the compiler does not have to be in the computer when the program is run since your program is converted to machine language. The effect of this is to give you much more usable memory. By compiling your programs and converting to a real-time interrupt-driven system, you could easily be playing Star Trek while your son, daughter, wife or husband is using the computer for help with their homework while the 6800 is controlling your heating and cooling system while etc., etc.

When the first reasonably priced compiler hits the personal computer market, all BASIC timing comparisions will have become totally meaningless. 6800 BASIC runs slow?

"Frankly, my dear, I don't care, I love my 6800."

LIST OF REFERENCES

. . . BASIC Timing Comparisions - Rugg, Feldman; Kilobaud, June 1977

. . . MITS BASIC, Poly BASIC, and NIBL - Raskin; Dr. Dobbs Journal of Computer Calistenics and Orthodontia, April 1977

Mickey A. Ferguson, WA4KDC
P.O. Box 708
Trenton, Georgia 30752

MEGABAUD*1.2, MAYBE?

Computers are great! You can explore the galaxy without leaving the corner of the utility room that your wife finally let you use for the funny set of boxes, play golf in air conditioned semi-comfort or be a ruler whose name you can't even pronounce.
Now . . . . . here's your chance to hot rod the SWTPC 6800 without getting yourr hands greasy.

Baud rates! Get your computer to cruise at a higher speed. First, make sure your terminal can operate at 1200 baud. If it is a TV-II you have it made. All the instructions are right there in those loose leaf papers they sent you with the UART board, happy hunting.

Remember when you built the computer serial interface board, the one that's plugged into port #1, if you ever got your system running. The people in S.A. (San Antonio not South Africa) gave you the option of 110 or 300 baud, and two jumpers with which to make your choice. You probably selected 300 baud. If it was 110 baud because you are using a teletype, didn't you notice that you were left out of this article by the algorithm in paragraph #2?

The jumper from point C actually selects the number of stop bits and not baud rate. A connection between C and 300 will be correct for any rate above 110, so we will leave this jumper alone. The connection from point D is the one that selects the rates.

The quick and dirty way for this mod is to clip the jumper where it enters the 300 hole and attach a test lead from that little wire to the 1200 b pin of an unused port. Perhaps you would like to use the 150 b pin to slow the thing down so you can read a long program while it is being listed. Just remember to adjust the terminal to the same rate or all you will get on your screen is garbage.

The neat method will be left up to you. One way to do it will be to mount a SP4T switch on the computer cabinet, connect the single pole to the hole at D and the 4 throws to the appropriate lands on the bottom of the mother board.

If you are using a cassette interface e.g. the AC-30 you will soon observe that you must switch back to 300 baud to dump programs. You can, however, load 300 baud tapes into the computer with it set on 1200 baud. This is due to the fact that KCS tapes are self-clocking. The terminal, however, will not respond to this self-clocking and if you are loading a BASIC program, a screen full of garbage will appear. On the same note, the terminal will not detect the character to stop the recorder at the end of the recorded program so you will have to turn it off yourself when the question marks cease.

Perhaps in the future we will discuss a slick method to automatically switch back to 300 baud during tape operation.

R. L. Hilbun
8109 Willow Creek Blvd.
Oklahoma City, Oklahoma 73132

MEMTST—A Memory Test Program

Errors produced by marginal RAM chips can be difficult to find if they are transient in nature, triggered by temperature, line voltage surges, noise spikes, etc. This program is an aid in locating the defective bit. The program tests the memory, and when it finds an error it writes its location on the control terminal and then resumes testing. Thus you can leave your computer running all day while you are at work, and when you come home you have a listing of bad memory locations and bits in error.

The program is entirely contained in the MIKBUG RAM starting at \$A014 and extending through \$A07D, jumping over the stack at \$A038 through \$A049. The starting address of the memory to be tested is loaded into \$A002-3 and then ending address in \$A004-5. The program starts by loading the B Accumulator with the contents of \$A000. The first memory location is loaded with the contents of B, then B is incremented and loaded into the next memory location and so on until the ending location is loaded. The program then returns to the first memory location and checks to see if the contents are the same as was loaded. The rest of the memory under test is checked in the same way. If a disagreement is found the program writes on the control terminal the incorrect memory content, the correct memory content, and the address of the erroneous byte. Otherwise nothing is written. Register \$A000 is then incremented and the process repeated. To keep you from worrying if the program is running if your memory is running error free, and thus no print out, the cursor is moved to the right by a space after loading the memory, and then moved back by a back space after testing. After 256 cycles all possible byte values have been tried in all of the memory locations under test. It then repeates again and again until you stop it with the RESET switch.

My system is a SWTPC M-6800 with 24K bytes of RAM, AC-30 Cassette Interface, CT-1024 TVT, and IBM 731 Selectric for hard copy output.

Emerson Brooks, D.Eng.
517 Melody Lane
Richardson, Texas 75081

```
PAGE  001    MEMTST

                          NAM     MEMTST

                          OPT     0,S

                     *TEST FOR MEMORY FAILURE.
                     *PUT START ADDRESS IN $A002, END IN $A004.
                     *PROGRAM LOADS SEQUENTIAL VALUES IN SUCCESSIVE
                     *LOCATIONS, THEN CHECKS.  IF CHECK IS INCORRECT
                     *INCORRECT VALUE, CORRECT VALUE AND LOCATION
                     *ARE PRINTED.  CURSOR IS MOVED BACK AND FORTH
                     *WITH EACH CYCLE TO SHOW THAT THE PROGRAM IS
                     *RUNNING WITH NO MEMORY ERRORS.

          A002       MEMBEG EQU    $A002    TEST START ADDRESS
          A004       MEMEND EQU    $A004    TEST END ADDRESS
          A000       BSTORE EQU    $A000    INITIAL LOAD VALUE
          A00E       WRONG  EQU    $A00E    INCORRECT VALUE
          A00F       RIGHT  EQU    $A00F    CORRECT VALUE
          A010       BADMEM EQU    $A010    ERROR ADDRESS
          000A       LF     EQU    $0A      LINE FEED
          000D       CR     EQU    $0D      CARRIAGE RETURN
          0008       BS     EQU    $08      BACK SPACE
          E0CC       OUTS   EQU    $E0CC    OUTPUT SPACE
          E1D1       OUTEEE EQU    $E1D1    PRINT CHARACTER
          E0CA       OUT2HS EQU    $E0CA    PRINT 2 HEX, SPACE
          E0C8       OUT4HS EQU    $E0C8    PRINT 4 HEX, SPACE

A014                        ORG    $A014

A014 F6 A000  START  LDA B  BSTORE   LOAD MEMORY
A017 FE A002         LDX    MEMBEG
A01A E7 00    LOOP1  STA B  0,X
A01C BC A004         CPX    MEMEND
A01F 27 04           BEQ    CHECK    END ADDRESS?
A021 08              INX
A022 5C              INC B
A023 20 F5           BRA    LOOP1

A025 BD E0CC  CHECK  JSR    OUTS     MOVE CURSOR
A028 F6 A000         LDA B  BSTORE   TEST MEMORY
A02B FE A002         LDX    MEMBEG
A02E A6 00    LOOP2  LDA A  0,X
A030 11              CBA
A031 26 27           BNE    ERROR    GO PRINT MESSAGE
A033 BC A004  RETURN CPX    MEMEND
A036 20 12           BRA    JUMP     JUMP OVER STACK

A04A                 ORG    $A04A

A04A 27 04    JUMP   BEQ    CYCLE    END ADDRESS?
A04C 08              INX
A04D 5C              INC B
A04E 20 DE           BRA    LOOP2
```

```
PAGE  002    MEMTST

00145 A050 86 08    CYCLE  LDA A  #BS       MOVE CURSOR BACK
00146 A052 BD E1D1         JSR    OUTEEE
00147 A055 7C A000         INC    BSTORE    INCR. INITIAL LOAD
00148 A058 20 BA           BRA    START     DO ANOTHER TEST

00149 A05A FF A010  ERROR  STX    BADMEM    PRINT ERROR MESSAGE
00150 A05D B7 A00E         STA A  WRONG
00151 A060 F7 A00F         STA B  RIGHT
00152 A063 86 0D           LDA A  #CR       CARRIAGE RETURN
00153 A065 BD E1D1         JSR    OUTEEE
00154 A068 86 0A           LDA A  #LF       LINE FEED
00155 A06A BD E1D1         JSR    OUTEEE
00156 A06D CE A00E         LDX    #WRONG    POINT TO DATA
00157 A070 BD E0CA         JSR    OUT2HS    PRINT INCORRECT VALUE
00158 A073 BD E0CA         JSR    OUT2HS    PRINT CORRECT VALUE
00159 A076 BD E0C8         JSR    OUT4HS    PRINT ERROR ADDRESS
00160 A079 FE A010         LDX    BADMEM    RESTORE INDEX
00161 A07C 20 B5           BRA    RETURN

00162 A048                 ORG    $A048

00163 A048 A014     VECTOR FDB    $A014


00166                      END
MEMBEG A002
MEMEND A004
BSTORE A000
WRONG  A00E
RIGHT  A00F
BADMEM A010
LF     000A
CR     000D
BS     0008
OUTS   E0CC
OUTEEE E1D1
OUT2HS E0CA
OUT4HS E0C8
START  A014
LOOP1  A01A
CHECK  A025
LOOP2  A02E
RETURN A033
JUMP   A04A
CYCLE  A050
ERROR  A05A
VECTOR A048

TOTAL ERRORS 00000
```

THE WA4KDC OS-II

OS-II© is a refined, shortened, and hopefully improved version of the original WA4KDC Operating System that was published in 73 Magazine. It should correct most of the problems that some people had with the earlier version and should be easier for most people to relocate, as we are publishing a complete listing of the program. Regardless of where in memory that you place OS-II, I would recommend that you place it in a block of memory separate from the remainder of your memory to keep other programs from overwriting it. This is because some programs (such as most BASIC's) clear all contiguous memory when initialized.

One of the most common complaints about the original WA4KDC OS was its extensive use of the Mikbug* RAM located at $A000-$A07F. It seems that everyone tries to use this 128 byte RAM for their own purposes and all sorts of programs end up fighting over it. OS-II makes minimal use of the Mikbug RAM and where OS-II does it, the RAM is used for the same purpose that Mikbug uses the location.

OS-II prompts with:
OS READY

At this point, OS-II is ready to accept your command. All commands are a single character in length, they are as follows:

B . . . Block move
C . . . Call (BASIC)
D . . . Dump
G . . . Goto
L . . . List
M . . . Mikbug
R . . . Read tape
W . . . Write tape
Z . . . Zero memory

Provision for an additional command (of your choice) has been included in OS-II. To implement an additional command, at the label "SPARE", insert the ASCII character you with to have OS-II recognize as the command to execute your function (any ASCII character other than those listed above may be used). In the following two bytes you should then place the address of the entry point of your function. Remember, your function **must** be written as a subroutine (end with an RTS) as all functions of the OS-II are implemented as subroutines.

Most functions require that you input additional information before they can execute, usually, the additional information is in the form of four digit hexidecimal addresses. When this is the case, OS-II will add a "$" in the space preceding your input as a reminder to you that it expects

you to input four hex digits. Should you type anything that is not a hex character, system control will be given to Mikbug.

My apologies to those who are using unmodified CT-1024's as OS-II is written for a scrolling type terminal and never issues a Home-up/Erase. You may, however, add this to the prompt string when you rea-semble OS-II for use in your system.

Now, let's look at the functions of OS-II. Let's begin with the Block Move function. The Block Move function does just that, it moves a block of data from anywhere in memory to anywhere in memory. It does not take care of any "housekeeping" (you will have to change any extended addresses) because it does not know whether it is moving a program or data. After each byte is moved, it is read at the new location and tested ot see if it is correct. If bad data is read at the new location, a software interrput will occur and the contents of the registers will be printed. Should this occur, the index register will contain the address of the location of the problem, the "A" reg. will contain the correct data, and the "B" reg. will contain the bad data. This should be helpful in locating and troubleshooting memory bugs! The Block Move function issues three prompts: "TO", "START", and "STOP". When is issues the prompt "TO", it is asking for the lowest address of the destination of the move. The "START" and "STOP" prompts are asking for the lowest and highest addresses of the block of data that you want moved.

The Call (Basic) function is really just an "automatic block move". And it could be used to call an editor, assembler, startrek or anything else you desire. The RUN, LBASIC, and HBASIC labels respectively define the "TO", "START", and "STOP" for block moving Basic from high memory (where it is stored) to low memory where it is executed. The values given in the listing assume Tom Pittman's Tiny Basic is stored at $3400 thru $3BFF. Let's assume that we have OS-II in memory and want to use Tom's Tiny as the Basic in the Call Basic function. How do we go about doing this? Assuming OS-II has a starting address of $3000, first load Tiny Basic into memory from tape and use the Block Move function of OS-II to move it to $3400-$3BFF. Then you make a tape beginning at $3400 thru the end of OS-II. Anytime thereafter that you load this tape, you will be loading both OS-II and Basic; with the Call Basic function automatically moving and executing Basic when it is used. The main advantage of having the Call Basic function is that you are able to alternate between running Basic and machine language programs (in low memory) without having to wait while Basic is reloaded from tape each time.

The Dump function of OS-II works a bit differently than the Dump function of the original WA4KDC OS. The Dump function of OS-II prints the memory dump on both the control terminal and on the PR-40 (turn the PR-40 off, if you don't want a permanent copy). And the Dump may be interrupted by tapping the"ESCape" key on the control terminal. The Dump function prompts with "START" and "STOP", in both cases it is asking for hexidecimal addresses and it prints a dump of everything between the two addresses that you give it.

The Goto function was added because I received a number of requests for it. The Goto function prompts with "TO" and wants an address. When you input an address it does a jump to subroutine to the address you have in memory and execute them in any order you wish.

The list function works very much as it did in the original WA4KDC OS, except it prints the listing on both the control terminal and the PR-40. It may be interrupted, as in the Dump function, by tapping the "ESCape" key on the control terminal. If you want a more complete disassembler, might I suggest mine - published in Kilobaud Magazine.

The Read Tape and Write Tape functions should be considered together. The Write function prompts with:

NAME?	(What's the name of the program?)
START?	(What's the lowest address of the program?)
STOP?	(What's the highest address of the program?)
PC?	(What's the entry point or initial program counter setting?)

When you have supplied all of this information, OS-II will make your tape. It begins with three seconds of nulls, then the name of the program, the program itself, the program counter, and finally an S9. By putting all of this information on the tape, you may fill your tapes with programs and use the Read Tape function to find the particular program you desire to load - and load it. When OS-II is outputting your program to tape, it uses a modified Mikbug format in that it outputs 240 bytes per line instead of the sixteen byte lines used by Mikbug. These tapes are directly loadable by Mikbug and save more than thirty seconds per kilobyte of memory put to tape. In either the Read Tape of Write Tape functions, when OS-II has issued the prompt "NAME?" you may input anything you desire as the name of the program (up to 64 characters). You terminate the input of the name with a carriage return. The Read function issues only the prompt

```
PAGE  001   OS-II

                           NAM     OS-II
          E0E3    MIKBUG   EQU     $E0E3
          E1D1    OUTEEE   EQU     $E1D1
          E1AC    INEEE    EQU     $E1AC
          E07E    PDATA1   EQU     $E07E
          E047    BADDR    EQU     $E047
          A048    PC       EQU     $A048
          E013    LOAD3    EQU     $E013
          E0BF    MOUT2H   EQU     $E0BF
          8004    CNTPIA   EQU     $8004
          801C    PTRPIA   EQU     $801C
                           OPT     O
3C00                       ORG     $3C00
3C00 8E A040      MAINLP   LDS     #$A040
3C03 BF A008               STS     $A008
3C06 CE 3C00               LDX     #MAINLP
3C09 FF A046               STX     PC-2
3C0C FF A048               STX     PC
3C0F 8D 1E        MAINL2   BSR     CRLF
3C11 CE 3F23               LDX     #READY
3C14 8D 1C                 BSR     PNTSTR
3C16 BD E1AC               JSR     INEEE
3C19 CE 3F59               LDX     #JMPTBL-
3C1C 08           MAINL3   INX
3C1D 08                    INX
3C1E 08                    INX
3C1F 8C 3F7A               CPX     #TBLEND
3C22 27 DC                 BEQ     MAINLP
3C24 A1 00                 CMP A   X
3C26 26 F4                 BNE     MAINL3
3C28 08                    INX
3C29 EE 00                 LDX     X
3C2B AD 00                 JSR     X
3C2D 20 D1                 BRA     MAINLP
3C2F CE 3F1E      CRLF     LDX     #CRSTR
3C32 7E E07E      PNTSTR   JMP     PDATA1
3C35 8D F8        BLOCK    BSR     CRLF
3C37 CE 3F31               LDX     #TO
3C3A 8D F6                 BSR     PNTSTR
3C3C BD E047               JSR     BADDR
3C3F FF 3F54               STX     TEMP1
3C42 8D EB                 BSR     CRLF
3C44 CE 3F36               LDX     #STARTS
3C47 8D E9                 BSR     PNTSTR
3C49 BD E047               JSR     BADDR
3C4C FF 3F56               STX     TEMP2
3C4F 8D DE                 BSR     CRLF
3C51 CE 3F3E               LDX     #STOPS
3C54 8D DC                 BSR     PNTSTR
3C56 BD E047               JSR     BADDR
3C59 FF 3F58               STX     TEMP3
3C5C 8D D1                 BSR     CRLF
3C5E FE 3F56      STRMOV   LDX     TEMP2
3C61 A6 00                 LDA A   X
3C63 BC 3F58               CPX     TEMP3
3C66 26 06                 BNE     MOVE1
3C68 FE 3F54               LDX     TEMP1
3C6B A7 00                 STA A   X
3C6D 39                    RTS
3C6E 08           MOVE1    INX
3C6F FF 3F55               STX     TEMP2
3C72 FE 3F54               LDX     TEMP1
3C75 A7 00                 STA A   X
3C77 E6 00                 LDA B   X
3C79 11                    CBA
3C7A 27 01                 BEQ     MOVE2
3C7C 3F                    SWI
3C7D 08           MOVE2    INX
3C7E FF 3F54               STX     TEMP1
3C81 20 DB                 BRA     STRMOV
3C83 8D AA        CBASIC   BSR     CRLF
3C85 CE 0100               LDX     #RUN
3C88 FF 3F54               STX     TEMP1
3C8B CE 3400               LDX     #LBASIC
3C8E FF 3F56               STX     TEMP2
3C91 CE 3BFF               LDX     #HBASIC
3C94 FF 3F58               STX     TEMP3
3C97 8D C5                 BSR     STRMOV
3C99 7E 0100               JMP     RUN
          0100    RUN      EQU     $100
          3400    LBASIC   EQU     $3400
          3BFF    HBASIC   EQU     $3BFF
3C9C 8D 91        DUMP     BSR     CRLF
3C9E CE 3F36               LDX     #STARTS
3CA1 BD E07E               JSR     PDATA1
3CA4 BD E047               JSR     BADDR
3CA7 FF 3F54               STX     TEMP1
3CAA 8D 83                 BSR     CRLF
3CAC CE 3F3E               LDX     #STOPS
3CAF BD E07E               JSR     PDATA1
3CB2 BD E047               JSR     BADDR
3CB5 FF 3F56               STX     TEMP2
3CB8 86 0D        DUMP1    LDA A   #13
3CBA 8D 4B                 BSR     OUTCHR
3CBC BD 3C2F               JSR     CRLF
3CBF CE 3F54               LDX     #TEMP1
3CC2 8D 1A                 BSR     OUT4HS
3CC4 C6 08                 LDA B   #8
3CC6 FE 3F54               LDX     TEMP1
3CC9 8D 15        DUMP2    BSR     OUT2HS
3CCB 09                    DEX
3CCC BC 3F56               CPX     TEMP2
3CCF 26 04                 BNE     DUMP3
3CD1 86 0D                 LDA A   #13
3CD3 20 32                 BRA     OUTCHR
3CD5 08           DUMP3    INX
3CD6 5A                    DEC B
3CD7 26 F0                 BNE     DUMP2
3CD9 FF 3F54               STX     TEMP1
3CDC 20 DA                 BRA     DUMP1
3CDE 8D 06        OUT4HS   BSR     OUT2H
3CE0 8D 04        OUT2HS   BSR     OUT2H
3CE2 86 20        OUTS     LDA A   #32
3CE4 20 21                 BRA     OUTCHR
3CE6 A6 00        OUT2H    LDA A   X
3CE8 8D 0F                 BSR     OUTHL
3CEA A6 00                 LDA A   X
3CEC 08                    INX
3CED 20 0E                 BRA     OUTHR
3CEF 8D 16        PDATA2   BSR     OUTCHR
3CF1 08                    INX
3CF2 A6 00        PDATA3   LDA A   X
3CF4 81 04                 CMP A   #4
3CF6 26 F7                 BNE     PDATA2
3CF8 39                    RTS
3CF9 44           OUTHL    LSR A
3CFA 44                    LSR A
3CFB 44                    LSR A
3CFC 44                    LSR A
3CFD 84 0F        OUTHR    AND A   #$F
3CFF 8B 30                 ADD A   #$30
3D01 81 39                 CMP A   #$39
3D03 23 02                 BLS     OUTCHR
3D05 8B 07                 ADD A   #7
3D07 8D 29        OUTCHR   BSR     BREAK
3D09 36                    PSH A
3D0A BD E1D1               JSR     OUTEEE
3D0D 32                    PUL A
3D0E FF 3F5A      OUTCH1   STX     XTEMP
3D11 37                    PSH B
3D12 CE 801C               LDX     #PTRPIA
3D15 C6 FF                 LDA B   #$FF
3D17 E7 00                 STA B   X
3D19 C6 3F                 LDA B   #$3F
3D1B E7 01                 STA B   1,X
3D1D A7 00                 STA A   X
3D1F C6 37                 LDA B   #$37
3D21 E7 01                 STA B   1,X
3D23 C6 3F                 LDA B   #$3F
3D25 E7 01                 STA B   1,X
3D27 6D 01        OUTCH2   TST     1,X
3D29 2A FC                 BPL     OUTCH2
3D2B E6 00                 LDA B   X
3D2D FE 3F5A               LDX     XTEMP
3D30 33                    PUL B
3D31 39                    RTS
3D32 36           BREAK    PSH A
3D33 B6 8004               LDA A   CNTPIA
3D36 2A 02                 BPL     BREAK1
3D38 32                    PUL A
3D39 39                    RTS
3D3A B6 8004      BREAK1   LDA A   CNTPIA
3D3D 2A FB                 BPL     BREAK1
3D3F 86 0D        BREAK2   LDA A   #$D
3D41 8D CB                 BSR     OUTCH1
3D43 BD 3C2F               JSR     CRLF
3D46 7E 3C00               JMP     MAINLP
3D49 BD 3C2F      LIST     JSR     CRLF
3D4C CE 3F36               LDX     #STARTS
3D4F BD E07E               JSR     PDATA1
```

"NAME?". When you have completed inputting the name of the program that you want loaded, the Read function will turn the tape reader on and begin searching for the program you have selected. When the desired program is found, it is loaded into memory and system control is given to Mikbug. From that point, typing"G" will execute the program. There a couple of points you should keep in mind when you are using the Read Tape function. If the program that you name is not on the tape, OS-II will keep searching to the end of the tape and you will have to push the "RESET" button to escape the read function or you may insert the proper cassette into the recorder and restart the recorder. Also, suppose you have a cassette in the recorder which has the following programs on it: Micro-Basic, Tiny Basic, 4K Basic, 8K Basic, and 12K Basic. If you tell OS-II to read "Basic" as the desired program, it will load the ifrst Basic it finds. If you wanted 8K Basic, you should have told it to read either "8K" or"8K Basic".

```
3D52 BD E047          JSR    BADDR
3D55 FF 3F54          STX    TEMP1
3D58 BD 3C2F          JSR    CRLF
3D5B CE 3F3E          LDX    #STOPS
3D5E BD E07E          JSR    PDATA1
3D61 BD E047          JSR    BADDR
3D64 FF 3F56          STX    TEMP2
3D67 CE 3F1E LIST2    LDX    #CRSTR
3D6A 8D 86            BSR    PDATA3
3D6C CE 3F54          LDX    #TEMP1
3D6F BD 3CDE          JSR    OUT4HS
3D72 FE 3F54          LDX    TEMP1
3D75 A6 00            LDA A  X
3D77 B7 3F58          STA A  TEMP3
3D7A BD 3CE0          JSR    OUT2HS
3D7D FF 3F54          STX    TEMP1
3D80 BD 3CE2          JSR    OUTS
3D83 5F               CLR B
3D84 B6 3F58          LDA A  TEMP3
3D87 81 8C            CMP A  #$8C
3D89 27 18            BEQ    LIST3
3D8B 81 8E            CMP A  #$8E
3D8D 27 14            BEQ    LIST3
3D8F 81 CE            CMP A  #$CE
3D91 27 10            BEQ    LIST3
3D93 84 F0            AND A  #$F0
3D95 81 20            CMP A  #$20
3D97 27 0B            BEQ    LIST4
3D99 81 60            CMP A  #$60
3D9B 25 08            BCS    LIST5
3D9D 84 30            AND A  #$30
3D9F 81 30            CMP A  #$30
3DA1 26 01            BNE    LIST4
3DA3 5C      LIST3    INC B
3DA4 5C      LIST4    INC B
3DA5 F7 3F59 LIST5    STA B  TEMP3+1
3DA8 27 10            BEQ    LIST8
3DAA 7A 3F59          DEC    TEMP3+1
3DAD 27 05            BEQ    LIST6
3DAF BD 3CDE          JSR    OUT4HS
3DB2 20 03            BRA    LIST7
3DB4 BD 3CE0 LIST6    JSR    OUT2HS
3DB7 FF 3F54 LIST7    STX    TEMP1
3DBA B6 3F54 LIST8    LDA A  TEMP1
3DBD B1 3F56          CMP A  TEMP2
3DC0 25 A5            BCS    LIST2
3DC2 B6 3F56          LDA A  TEMP2
3DC5 B1 3F54          CMP A  TEMP1
3DC8 25 08            BCS    JMPBRK
3DCA B6 3F55          LDA A  TEMP1+1
3DCD B1 3F57          CMP A  TEMP2+1
3DD0 25 95            BCS    LIST2
3DD2 7E 3D3F JMPBRK   JMP    BREAK2
3DD5 BD 3C2F GOTO     JSR    CRLF
3DD8 CE 3F31          LDX    #TO
3DDB BD E07E          JSR    PDATA1
3DDE BD E047          JSR    BADDR
3DE1 6E 00            JMP    X
3DE3 BD 3C2F READ     JSR    CRLF
3DE6 CE 3F4D          LDX    #NAMSTR
3DE9 BD E07E          JSR    PDATA1
3DEC 8D 27            BSR    LNAME
3DEE BD 3C2F          JSR    CRLF
3DF1 86 3C            LDA A  #$3C
3DF3 B7 8007          STA A  CNTPIA+3
3DF6 86 11            LDA A  #$11
3DF8 BD E1D1          JSR    OUTEEE
3DFB CE 3F7E READ2    LDX    #NAMSTO
3DFE BD E1AC READ3    JSR    INEEE
3E01 A1 00            CMP A  X
3E03 26 F6            BNE    READ2
3E05 08               INX
3E06 A6 00            LDA A  X
3E08 81 04            CMP A  #4
3E0A 26 F2            BNE    READ3
3E0C 8E A042          LDS    #$A042
3E0F BF A008          STS    $A008
3E12 7E E013          JMP    LOAD3
3E15 CE 3F7E LNAME    LDX    #NAMSTO
3E18 BD E1AC LNAME2   JSR    INEEE
3E1B A7 00            STA A  X
3E1D 08               INX
3E1E 81 0D            CMP A  #$D
3E20 26 F6            BNE    LNAME2
3E22 09               DEX
3E23 86 04            LDA A  #4
3E25 A7 00            STA A  X
3E27 39               RTS
3E28 BD 3C2F WRITE    JSR    CRLF
3E2B CE 3F4D          LDX    #NAMSTR
3E2E BD E07E          JSR    PDATA1
3E31 8D E2            BSR    LNAME
3E33 BD 3C2F          JSR    CRLF
3E36 CE 3F36          LDX    #STARTS
3E39 BD E07E          JSR    PDATA1
3E3C BD E047          JSR    BADDR
3E3F FF 3F54          STX    TEMP1
3E42 BD 3C2F          JSR    CRLF
3E45 CE 3F3E          LDX    #STOPS
3E48 BD E07E          JSR    PDATA1
3E4B BD E047          JSR    BADDR
3E4E FF 3F56          STX    TEMP2
3E51 BD 3C2F          JSR    CRLF
3E54 CE 3F48          LDX    #PCSTR
3E57 BD E07E          JSR    PDATA1
3E5A BD E047          JSR    BADDR
3E5D FF A048          STX    PC
3E60 86 12            LDA A  #$12
3E62 BD E1D1          JSR    OUTEEE
3E65 8D 23            BSR    OUT00
3E67 CE 3F7A          LDX    #TBLEND
3E6A BD E07E          JSR    PDATA1
3E6D 8D 24            BSR    TPUNCH
3E6F CE A048          LDX    #PC
3E72 FF 3F54          STX    TEMP1
3E75 08               INX
3E76 FF 3F56          STX    TEMP2
3E79 8D 18            BSR    TPUNCH
3E7B BD 3C2F          JSR    CRLF
3E7E CE 3F45          LDX    #S9STR
3E81 BD E07E          JSR    PDATA1
3E84 86 14            LDA A  #$14
3E86 BD E1D1          JSR    OUTEEE
3E89 39               RTS
3E8A C6 5A   OUT00    LDA B  #90
3E8C BD E1D1 OUT01    JSR    OUTEEE
3E8F 5A               DEC B
3E90 26 FA            BNE    OUT01
3E92 39               RTS
3E93 FE 3F54 TPUNCH   LDX    TEMP1
3E96 FF A00F          STX    $A00F
3E99 B6 3F57 PUN11    LDA A  TEMP2+1
3E9C B0 A010          SUB A  $A010
3E9F F6 3F56          LDA B  TEMP2
3EA2 F2 A00F          SBC B  $A00F
3EA5 26 04            BNE    PUN22
3EA7 81 F0            CMP A  #$F0
3EA9 25 02            BCS    PUN23
3EAB 86 EF   PUN22    LDA A  #$EF
3EAD 8B 04   PUN23    ADD A  #4
3EAF B7 A011          STA A  $A011
3EB2 80 03            SUB A  #3
3EB4 B7 A00E          STA A  $A00E
3EB7 CE E134          LDX    #$E134
3EBA BD E07E          JSR    PDATA1
3EBD 5F               CLR B
3EBE CE A011          LDX    #$A011
3EC1 8D 24            BSR    PUNT2
3EC3 CE A00F          LDX    #$A00F
3EC6 8D 1F            BSR    PUNT2
3EC8 8D 1D            BSR    PUNT2
3ECA FE A00F          LDX    $A00F
3ECD 8D 18   PUN32    BSR    PUNT2
3ECF 7A A00E          DEC    $A00E
3ED2 26 F9            BNE    PUN32
3ED4 FF A00F          STX    $A00F
3ED7 53               COM B
3ED8 37               PSH B
3ED9 30               TSX
3EDA 8D 0B            BSR    PUNT2
3EDC 33               PUL B
3EDD FE A00F          LDX    $A00F
3EE0 09               DEX
3EE1 BC 3F56          CPX    TEMP2
3EE4 26 B3            BNE    PUN11
3EE6 39               RTS
3EE7 EB 00   PUNT2    ADD B  X
3EE9 7E E0BF          JMP    MOUT2H
3EEC BD 3C2F ZERO     JSR    CRLF
3EEF CE 3F36          LDX    #STARTS
3EF2 BD E07E          JSR    PDATA1
3EF5 BD E047          JSR    BADDR
3EF8 FF 3F54          STX    TEMP1
3EFB BD 3C2F          JSR    CRLF
3EFE CE 3F3E          LDX    #STOPS
3F01 BD E07E          JSR    PDATA1
3F04 BD E047          JSR    BADDR
3F07 FF 3F56          STX    TEMP2
3F0A FE 3F54          LDX    TEMP1
3F0D 4F               CLR A
3F0E 6F 00   CLEAR    CLR    X
3F10 A1 00            CMP A  X
3F12 27 01            BEQ    CLEAR2
3F14 3F               SWI
3F15 08      CLEAR2   INX
3F16 BC 3F56          CPX    TEMP2
3F19 26 F3            BNE    CLEAR
3F1B 6F 00            CLR    X
3F1D 39               RTS
```

The Zero Memory function will clear any block of memory that you specify. If the block specified in non-existent, ROM, or if you have a problem with the RAM such that it will not clear; you will get a software interrupt as explained in the discussion on the Block Move function earlier. The Zero Memory function can be very useful in testing tapes, clearing buffers, etc.

I almost forgot to tell you about the Mikbug function! This turns system control to Mikbug. From this point you may return to OS-II by typing "G".

To the best of my knowledge, this is the final operating system that you will see from this author. It is, I believe, thoroughly debugged and in it's final form. It is very useful and powerful; but all things of this sort tend to be rather timely. A year or two from now we may all be laughing at any non-disc-based systems software, but for now, use and enjoy!

73's & Happy Computing!

Mikey Ferguson, WA4KDC
P.O. Box 708
Trenton, Georgia 30752

```
03600 3F1E 0D       CRSTR  FCB   $D,$A,0,0,4
      3F1F 0A
      3F20 00
      3F21 00
      3F22 04
03610 3F23 4F       READY  FCC   /OS READY/
      3F24 53
      3F25 20
      3F26 52
      3F27 45
      3F28 41
      3F29 44
      3F2A 59
03620 3F2B 0D              FCB   $D,$A,0,0,62,4
      3F2C 0A
      3F2D 00
      3F2E 00
      3F2F 3E
      3F30 04
03630 3F31 54       TO     FCC   /TO $/
      3F32 4F
      3F33 20
      3F34 24
03640 3F35 04              FCB   4
03650 3F36 53       STARTS FCC   /START $/
      3F37 54
      3F38 41
      3F39 52
      3F3A 54
      3F3B 20
      3F3C 24
03660 3F3D 04              FCB   4
03670 3F3E 53       STOPS  FCC   /STOP $/
      3F3F 54
      3F40 4F
      3F41 50
      3F42 20
      3F43 24
03680 3F44 04              FCB   4
03690 3F45 53       S9STR  FCC   /S9/
      3F46 39
03700 3F47 04              FCB   4
03710 3F48 50       PCSTR  FCC   /PC $/
      3F49 43
      3F4A 20
      3F4B 24
03720 3F4C 04              FCB   4
03730 3F4D 4E       NAMSTR FCC   /NAME? /
      3F4E 41
      3F4F 4D
      3F50 45
      3F51 3F
      3F52 20
03740 3F53 04              FCB   4
03750 3F54 0002     TEMP1  RMB   2
03760 3F56 0002     TEMP2  RMB   2
03770 3F58 0002     TEMP3  RMB   2
03780 3F5A 0002     XTEMP  RMB   2
03790 3F5C 42       JMPTBL FCB   66
03800 3F5D 3C35            FDB   BLOCK
03810 3F5F 43              FCB   67
03820 3F60 3C83            FDB   CBASIC
03830 3F62 44              FCB   68
03840 3F63 3C9C            FDB   DUMP
03850 3F65 47              FCB   71
03860 3F66 3DD5            FDB   GOTO
03870 3F68 4C              FCB   76
03880 3F69 3D49            FDB   LIST
03890 3F6B 4D              FCB   77
03900 3F6C E0E3            FDB   MIKBUG
03910 3F6E 52              FCB   82
03920 3F6F 3DE3            FDB   READ
03930 3F71 57              FCB   87
03940 3F72 3E28            FDB   WRITE
03950 3F74 5A              FCB   90
03960 3F75 3EEC            FDB   ZERO
03970 3F77 0003     SPARE  RMB   3
03980 3F7A 0D       TBLEND FCB   $D,$A,0,0
      3F7B 0A
      3F7C 00
      3F7D 00
03990 3F7E 0040     NAMSTO RMB   64
04000 A048                 ORG   PC
04010 A048 3C00     PC     FDB   MAINLP
04020                      END

TOTAL ERRORS 00000



ENTER PASS : 1P,2P,2L,2T
```

Editorial Comments: Thank you Mickey for your fine contributions to both the ICCD and the computer users—we appreciate your work.